N88-16400

The Use and Generation of Illustrative Examples in Computer-Based Instructional Systems

William John Selig
NASA/MSFC

and

James D. Johannes
The University of Alabama in Huntsville

ABSTRACT

Examples are both pervasive and necessary in the teaching of new material. One of the more common types is the illustrative example, used to clarify and instantiate general statements. The use of illustrative examples in computer-based instructional systems to date, when they have been used at all, has been generally limited to some form of 'canned' text. This method has the problem that as the system evolves, the canned supportive material does not necessarily follow along. This paper proposes a method whereby the underlying domain knowledge is represented such that illustrative examples may be generated on demand. This method has the advantage that the generated example can follow changes in the domain in addition to allowing automatic customization of the example to the individual.

1. Introduction

As the human race continues its venture into space, the supportive and operational systems necessary to accomplish this goal become increasingly more complex. It is becoming ever harder for an individual to grasp the overall function of the systems involved and thus it is becoming ever harder to effectively use these systems. Owing to both the growing complexity and size of our space effort, the need for efficient training systems is becoming critical. Computer based instructional systems, while still in various levels of development, hold out hope for just such efficiency of training due to their ability to tailor themselves to individuals' needs and their ability to time-share among many users [6,8]. In these instructional systems, examples will play an important role in increasing the efficiency of learning.

Examples have long been used in regular instructional materials to facilitate learning in a number of ways. Initial examples introduce material and pique user interest, thus motivating the learning experience. Application examples anchor the learned material in a wider framework, thus increasing retention and understanding. Evidential examples and control examples are used to support or test various instructional statements. Finally, illustrative examples and counterexamples are used to both clarify and define the limits of applicability of general statements [2]. This paper is specifically concerned with the use and generation of these illustrative examples, hereafter referred to as just 'examples'.

Current approaches to the use of examples in computer-based instructional systems involve some form of presupplied text to determine what to say. These cover the range from 'canned' text to templates to feature scripts [1]. Alternatively, the system may contain an internal expert to generate an example of a 'better way', as in some coaching systems [9]. The problem with all of these systems, however, is that they depend upon someone deciding ahead of time what an appropriate example is for some given concept. In some domains, such as mathematics, this is adequate. In other domains, such as spacecraft or computer system operations, the ability to tailor the example to the user and his working environment is potentially more useful. This paper proposes just such a method.

This method of example generation uses a knowledge representation scheme based upon Sowa's conceptual graph theory. In this scheme there exists what Sowa calls a semantic net, similar to a cross-linked hierarchy, in which the basic interrelationships among the primitive concepts in the domain is captured. Domain knowledge about higher level concepts and actual entities is represented by conceptual graphs, similar to Schank's conceptual dependency theory representation, but with a potentially unlimited set of relations [5,7]. To generate an example from this domain representation, the concept to be exemplified is also represented as a conceptual graph with various attributes and parameters. This graph is then joined over the domain representation to fill in the missing pieces, either through full or partial matching on existing knowledge or generation based upon domain first principles.

The most applicable related research is that of Edwina Rissland on constrained example generation (CEG), in which an example is generated from an examples knowledge base using prespecified constraints. CEG lists three methods of 'generation'; retrieval, modification and construction. In retrieval and modification, pre-existing examples are used, either directly or with modification, that have been precompiled by the system builder. It is only in the construction phase that an example is actually generated from domain first principles, and even here the construction may instead be done by combining existing examples to create a larger example. The work on CEG is currently focused only on the retrieval and modification aspects of this method and has yet to address the construction aspect of generation from domain first principles [4]. This paper presents a method which includes that aspect.

2. Methodology

A typical situation where instructional systems would be useful in space related operations involves the learning of a command and control system. Learning these systems is similar to learning a computer operating system. There are numerous commands with various effects, possibly different in different contexts, and during the use of these commands the user must have an understanding of the overall view of things. Because of this similarity, and for practical testing reasons, this paper presents the proposed method in the domain of instruction about the UNIX operating system.

Appendix A shows a portion of the semantic net of basic domain primitive concepts. These form a framework for talking about types of objects. Note that some of these types, such as entity and information, are domain independent while others, such as command, are domain dependent.

Appendix B shows portions of the set of conceptual graphs comprising the extensional knowledge about the domain [3] and the user environment. It is there that methods would be represented for obtaining information about actual files and for interpreting that information, about a directory for example.

The best way to illustrate the proposed method is, of course, with some examples. In these examples the concept to be exemplified is first presented textually as might be composed by an instructional designer. Next is presented a conceptual graph for that concept, followed by a description of the processing involved and the resultant exemplifying conceptual graph.

Example 1

"The UNIX Programmer's Manual is kept on-line. You can use the 'man' command to print the manual pages for a system command."

```
COMMAND
        COMMAND-NAME: man
        COMMAND-ARGUMENT: *command-name
```

This conceptual graph (CG) would be matched on the command name field to the CG for the 'man' command (refer to Appendix B). Since no options are specified in the example CG, they would not be included in the match result. This resultant CG would specify that the command argument must be in the intersection of the set of manual titles and the set of command names. An entry from that intersection would be randomly chosen and inserted into the command argument field resulting in the following exemplifying graph:

```
COMMAND
        COMMAND-NAME: man
        COMMAND-ARGUMENT: sort
```

which an English generator would render as "man sort".

Example 2

The previous example was relatively simple. A more complicated example involves the explanation of the use of shell meta-characters.

"The * can be used in conjunction with ?, as in ??b*, to match multiple filenames."

```
SET
     SET-SIZE: >=2
     SET-ELEMENTS:
          FILE-NAME
               LENGTH: >=3
               PATTERN: ??b*
```

First we match this CG with the FILE-NAME CG in the semantic net. This sets the length field to 3-14. At this point we can either match against the user environment or generate filenames directly. If we choose to match against the user environment, we would call a routine to return all filenames in the user's current directory and attempt to match those against the requirements. If we matched enough of those filenames, we would return that set. Alternatively, generating the filenames would involve instantiating patterns that met both the length and pattern constraints. In either case an actor CG for matching would be invoked which knew about character patterns and meta-characters.

A generated result (compressed for brevity) might be

```
SET
     SET-SIZE: 4
     SET-ELEMENTS:
          FILE-NAME: { aab, 23bso, aabredor, debar }
```

3. Conclusions and Further Directions

This outlines the concept of system-generated examples and a method for achieving them. The present work has focused only on the representational and generational problems. Before generated examples can be fully used in computer-based instructional systems, some manner of generating English from the resultant conceptual graph is also needed. Additionally, it would be convenient to have a parser generate the initial conceptual graph from the English statement of the concept to be exemplified. However, these are separate and further areas for research. That examples are useful in regular instructional materials cannot be denied. Further work will allow them to benefit computer-based instructional systems as well.

REFERENCES

1) Bienkowski, M.A., R.E. Cullingford and M.W. Krueger, Generating Natural Language Explanations in a Computer-Aided Design System, Technical Report CS83-1, Laboratory of Computer Science Research, The University of Connecticut, 1983

2) Mandl, H., W. Schnotz and S. Tergan, On the Function of Examples in Instructional Texts, Presented at the 1984 AERA Annual Meeting

3) McGilton, H. and R. Morgan, Introducing the UNIX System, McGraw-Hill, 1983

4) Rissland, E.L., Constrained Example Generation, COINS Technical Report 81-24, University of Massachusetts at Amherst

5) Schank, R.G. and C.J Rieger III, Inference and the Computer Understanding of Natural Language, in Readings in Knowledge Representation, R.J. Brachman and H.J. Levesque (eds), Morgan Kaufman, 1985, pp. 119-139

6) Sinnot, L.T., Generative Computer-Assisted Instruction and Artificial Intelligence, Educational Testing Service, October, 1976

7) Sowa, J.F., Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984

8) Barr, A. and E.A. Feigenbaum (eds), The Handbook of Artificial Intelligence, V2, William Kaufman, 1982, pp. 225-8

9) Barr, A. and E.A. Feigenbaum (eds), The Handbook of Artificial Intelligence, V2, William Kaufman, 1982, pp. 254-60

APPENDIX A - Intensional Knowledge

This is a simplified representation showing only the concept types and not the relations linking the concepts involved. All relation types in this example are characteristics and are indicated by indentation.

```
ENTITY < (is a subtype of)  UST (the universal subtype)
     includes physical objects as well as abstractions

INFORMATION < UST        Anything that can be communicated


FILE-NAME < ENTITY, INFORMATION    The entity used to access a file
     LENGTH: 1-14
     PATTERN: { valid-characters }

COMMAND-NAME < FILE-NAME

COMMAND-DESCRIPTION < TEXT

COMMAND-OPTION < INFORMATION
     OPTION-LETTER
     OPTION-DESCRIPTION
     OPTION-ARGUMENT

COMMAND < ENTITY                   Performs an operation
     COMMAND-NAME
     COMMAND-OPTION
     COMMAND-ARGUMENT
     COMMAND-DESCRIPTION
```

APPENDIX B - Extensional Knowledge

```
COMMAND
     COMMAND-NAME:
          LENGTH: 3
          PATTERN: man
     COMMAND-DESCRIPTION: "finds information by keywords;  prints
                          selected manual pages"
     COMMAND-OPTION: {
          OPTION-LETTER: k
               -DESCRIPTION: "prints  a 1-line synopsis of  each
                             manual  section whose  listing
                             in   the  table  of   contents
                             contains one of the keywords"
               -ARGUMENT: { keywords },
          OPTION-LETTER: t
               -DESCRIPTION: "forces use of TROFF format"
                    }
     COMMAND-ARGUMENT: { manual-titles }

manual-titles: {at,awk,cat,cc, ... ls,man, ... sort,tail,wc }
```